Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets

Moe Thandar Wynn[1], David Edmond[1], W.M.P. van der Aalst[1,2] and A.H.M. ter Hofstede[1]

[1] Center for IT Innovation, Queensland University of Technology
P.O. Box 2434, Brisbane Qld 4001, Australia.
{m.wynn,d.edmond,a.terhofstede}@qut.edu.au
[2] Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NIL-5600 MB, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tm.tue.nl

Abstract. Workflow languages offer constructs for coordinating tasks. Among these constructs are various types of splits and joins. One type of join, which shows up in various incarnations, is the OR-join. Different approaches assign a different (often only intuitive) semantics to this type of join, though they do share the common theme that synchronisation is only to be performed for active threads. Depending on context assumptions this behaviour may be relatively easy to deal with, though in general its semantics is complicated, both from a definition point of view (in terms of formally capturing a desired intuitive semantics) and from a computational point of view (how does one determine whether an OR-join is enabled?). In this paper the concept of OR-join is examined in detail in the context of the workflow language YAWL, a powerful workflow language designed to support a collection of workflow patterns and inspired by Petri nets. The OR-join's definition is adapted from an earlier proposal and an algorithmic approach towards determining OR-join enablement is examined. This approach exploits a link that is proposed between YAWL and Reset nets, a variant of Petri nets with a special type of arc that can remove all tokens from a place.

Keywords: OR-join, YAWL, Workflow patterns, synchronizing merge, Petri nets, Reset nets.

1 Introduction

Workflow specifications should capture various aspects of business models such as the flow of control, the flow of data, the structure of the organisation, and the use of resources (see e.g.[13]). The control flow perspective captures the execution interdependencies between the tasks of a business process. In-depth analysis and comparison of a number of commercially available workflow management systems has been performed [4]. The findings demonstrate that the interpretation of even the basic control flow constructs is not uniform and it is often unclear how the more complex requirements could be supported. The authors propose 20 workflow patterns to address control flow requirements in a language independent style. YAWL (Yet Another Workflow Language) is a result of this analysis, it provides direct support for most patterns [3]. YAWL has a formal semantics specified as a transition system. Although YAWL exploits concepts from Petri nets, it also provides direct support for those patterns hard to

realise in Petri nets. One of these patterns corresponds to the synchronising merge or the OR-join, the focus of this paper. In practice, there is a need for a construct like the OR-join as is evident from e.g. the fact that some commercial systems support OR-join like constructs. However, experience with these systems shows that it is difficult to select a suitable semantics and implement it efficiently. Workflow management systems like InConcert, eProcess, and WebSphere MQ Workflow have solved problems related to the OR-join using syntactical restrictions. IBM WebSphere MQ Workflow [17] (formerly known as MQSeries Workflow and FlowMark and also used as a basis for the new BPEL standard) offers full support for the OR-join but in order to do this it requires the workflow to be acyclic, i.e., the only way to introduce loops is by executing the entire (sub)process [2]. Other systems like Eastman and Domino Workflow seem to use a non-local semantics similar to the one used in YAWL. Such a non-local semantics may lead to unexpected results. Moreover, a non-local semantics may result in poor performance as is stated in the manual of Eastman: "Parallel instances can accumulate at a Join workstep if the instances are routed to the workstep by preprocessing rules. These instances will eventually be joined by a RouteEngine subprocess (thread) that examines Join worksteps for such instances. This Join scavenger thread reduces system efficiency, so routing to Join worksteps using preprocessing rules should be avoided" [9]. These examples illustrate the practical relevance of the OR-join and serve as a motivation for the work reported in this paper. For a more complete discussion on workflow systems' support for OR-join semantics, we refer to [2, 4, 14, 15].

The OR-join is a control flow construct that sometimes behaves like an AND join and sometimes like an XOR join based on the current context. Variants and interpretations of the OR-join have been proposed in the literature. In [18], several possible interpretations of OR-join semantics in the context of Event-driven Process Chains (EPCs) are discussed. If there is a matching OR-split, the OR-join semantics is taken to be "wait for the completion of all paths activated by the matching split". If there is no matching split, there could be at least three interpretations of an OR join: wait-for-all, first-come and every-time [18]. In [2], the authors highlight the technical, conceptual and practical problems with the formal semantics of the OR-join in Event driven Process Chains (EPCs). The authors suggest that there is no sound formal semantics for EPCs that is fully compliant with the informal semantics and that any formal semantics for EPCs will impose some restrictions or will deviate from the informal semantics to some extent. The authors demonstrate the problems using vicious circles, which are formed when two or more OR-joins are in a feedback loop and each OR-join waits for the other OR-join to complete first. On the other hand, in [15] a semantic framework for formally defining the non-local semantics of EPCs including the OR-join is proposed. The author states that "a single transition relation cannot precisely capture the informal semantics of EPCs". It is proposed that the non-local semantics be defined as a pair of transition relations and a semantic definition using techniques from fixed point theory is presented [15]. The current OR-join approach in YAWL [3] is intended to be a generalised approach and the formal semantics of the OR-join is defined by ignoring all other OR-joins. This approach is described as "ad hoc in some way" [15].

The contributions of this paper are threefold. Firstly, we re-examine the OR-join semantics as proposed in [3], because its behaviour is non-intuitive in the context of

OR-joins depending on other OR-joins and composite tasks (they cannot be treated like black boxes). Secondly, for the purposes of the OR-join definition and analysis, we propose an abstract view on YAWL, one which is formalised in terms of *Reset nets* [5–8, 10–12]. Reset nets are considered the most suitable formalism as reset arcs provide direct support for the cancellation feature in YAWL (another concept introduced to YAWL as a result of the workflow patterns and the difficulty of realising this feature in Petri nets). Thirdly, the mapping of YAWL nets to Reset nets is exploited to find an algorithmic solution to the non-trivial problem of OR-join enablement. Note that the contribution of this paper is not limited to YAWL. Many systems and languages struggle with the semantics and implementation of the OR-join. This paper provides suitable semantics and gives a concrete algorithm to support an efficient implementation.

This rest of the paper is organised as follows. In Section 2, we introduce the current OR-join semantics in YAWL, discuss the problems with this semantics and propose alternative treatments for OR-joins depending on other OR-joins in a YAWL net. In Section 3, the definitions of EWF-nets (Extended Workflow Nets) and Reset nets are presented together with the proposed abstractions to enable EWF-net to Reset net mappings. In Section 4, we propose a new semantics for the OR-join in YAWL. In Section 5, we propose an algorithm for OR-join analysis based on well-known backwards search techniques. Section 6 concludes the paper.

2 Current semantics of the OR-join in YAWL

In this section, we first outline the challenges associated with the non-local semantics of the OR-join. In particular, we show how ignoring other OR-joins during the analysis can lead to counter-intuitive results. We then propose some alternative treatments for OR-joins on the path to other OR-joins.

2.1 The OR-join in YAWL

A YAWL model is made up of tasks, conditions and a flow relation between tasks and conditions. In YAWL, tasks may be directly connected graphically. The splits, joins, conditions and cancellation symbols for YAWL are shown in Figure 1. YAWL uses the terms tasks and conditions to avoid confusion with Petri net terminology (transitions and places). If there is a cancellation set associated with a task, the execution of the task removes all the tokens from the conditions and tasks in the cancellation set. Cancelling a task is achieved by removing tokens from internal conditions of the task. An OR-join task is enabled at a marking iff at least one of its input conditions is marked and it is not

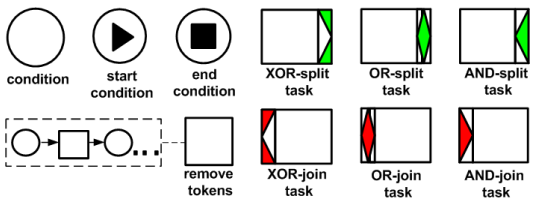

Fig. 1. Splits, joins, conditions and cancellation in YAWL

possible to reach a marking that marks all currently marked input conditions (possibly with fewer tokens) and at least one that is currently unmarked. If it is possible to place tokens in the unmarked input conditions of an OR-join in the reachable markings from the current marking, then the OR-join task should not be enabled and wait until either more input conditions are marked or until it is no longer possible to mark more input conditions.

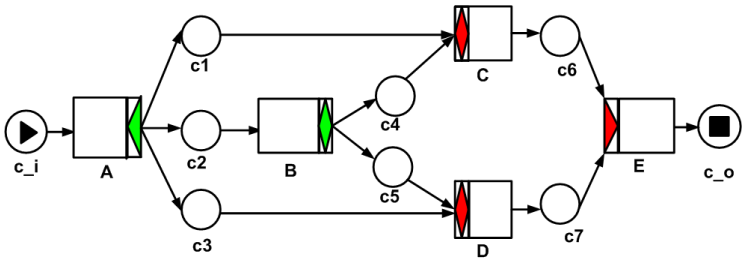

Fig. 2. A YAWL net with an OR-split task B and two OR-join tasks C and D

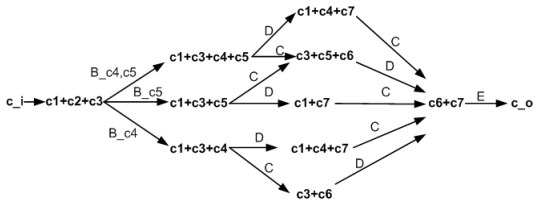

Fig. 3. Reachability graph of the YAWL net in Figure 2 (assuming some OR-join behaviour)

The example in Figure 2 demonstrates an unstructured YAWL net with AND-split task A, AND-join task E, OR-split task B and OR-join tasks C and D. This example demonstrates the different behaviours of OR-joins in the context of two different markings. First consider a marking $M = c1+c2+c3$ where there is a token in input condition $c1$ of OR-join task C and in input condition $c3$ of OR-join task D. To determine whether tasks C and/or D should be enabled at M, we need to find out whether tokens could be put into $c4$ or $c5$ in the reachable markings from M. The reachability graph of Figure 3 shows the reachable markings from the initial marking $M_0 = c_i$ to the end marking $M = c_o$.[1] We can see that by executing task B, we can reach markings $c1 + c3 + c5$ or $c1 + c3 + c4 + c5$ that mark $c5$, an unmarked input condition of task D in M. Also, markings $c1 + c3 + c4$, $c1 + c3 + c4 + c5$ could be reached by executing task B and they mark $c4$, an unmarked input condition of task C in M. As we can reach a new marking from M which can put a token in an unmarked input condition of the OR-join tasks C and D, neither task C or D should be enabled at M. If we consider a marking $M' = c1+c3+c4$, where all the input conditions of C (i.e., $c1$ and $c4$) are marked, then

[1] Note the overloading of notation, i.e., here c_o is a multiset denoting the marking with one token in condition c_o.

C would be enabled at M'. We will also enable task D at M' as it is not possible for another token to arrive at input condition $c5$. Note that in the scenario where we move from M to M', task D was not enabled in M and, although no tokens were added to the input conditions of this task, it got enabled in M'.

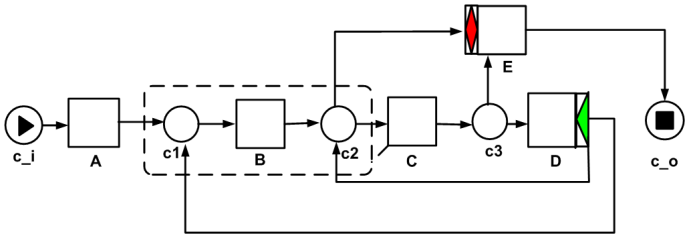

Fig. 4. Cancellation task C with an infinite loop

Now, let us consider OR-joins in the light of cancellation. In Figure 4, we describe a YAWL net with (i) task C removing tokens from the conditions $c1$, $c2$ and task B when firing, and (ii) an OR-join task E. At a marking $M = c2$, we marked one of the input conditions of E and we need to perform an analysis to decide whether both $c2$ and $c3$ could be marked in a reachable marking from M. We can observe the following sequence of reachable markings from M: $c2 \rightarrow^C c3 \rightarrow^D c1 + c2 \rightarrow^B 2c2 \rightarrow^C c3$. This is due to the cancellation feature of C, removing tokens from $c2$ when firing. We can conclude that it is not possible to reach a bigger marking $c2 + c3$ from M and therefore, E should be enabled at M. Let us consider a different situation where task C does not have a cancellation set associated with it. From marking $M = c2$, we can observe the following sequence of reachable markings: $c2 \rightarrow^C c3 \rightarrow^D c1 + c2 \rightarrow^B 2c2 \rightarrow^C c2+c3$. As we can reach $c2+c3$ which marks more input places of the OR-join task E, the analysis will conclude that task E should not be enabled at M. This example demonstrates the possible effect that the cancellation feature of a task can have on the OR-join enablement analysis.

From the above examples, it is obvious that the OR-join semantics requires careful analysis and the decision to enable an OR-join cannot be made locally. Any OR-join algorithm must evaluate all the reachable markings from a current marking to determine whether there is a possibility of a token arriving at an input condition of an OR-join which is not currently marked (while all input conditions which were already marked remain marked though possibly with fewer tokens). This algorithm potentially needs to be applied every time the marking changes and the OR-join analysis could place a significant load on any workflow engine required to execute it, cf. the quote from the manual of Eastman [9] in the introduction.

2.2 Problems with current OR-join semantics in YAWL

Two problems may be identified with the current OR-join semantics of YAWL which are related to the treatment of OR-joins and composite tasks preceding an OR-join under consideration.

The current OR-join semantics ignores other OR-joins when analysing whether a particular OR-join should be enabled at a given marking [3]. In Figure 5, there are two OR-join tasks, E and F in the YAWL net. Consider a marking $M = c1 + c3$ where the analysis for the OR-join task, F is performed. After executing task C, it is possible to reach either $c3 + c4$, $c3 + c5$ or $c3 + c4 + c5$. One possible occurrence sequence is $c1 + c3 \rightarrow^C c3 + c4 + c5 \rightarrow^D c3 + c4 + c6 \rightarrow^E c3 + c7$. Hence, $M' = c3 + c7$ is a reachable marking from M. However, the current OR-join semantics ignores other OR-joins on the path to F, so task E and the associated conditions will not be taken into account, and M' is therefore not considered as a reachable marking during the OR-join analysis of F. As a result, the analysis will conclude that there is no possibility of another token arriving in $c7$ and F would be enabled at M. This behaviour is probably not what one would expect from this specification. It could also result in multiple executions of task F and more than one token could be produced for c_o e.g. $(c1 + c_o)$. A YAWL model which can produce a token for the output condition c_o while still having tokens in the other conditions is considered as not having proper completion and is therefore not sound [1]. We have seen that as the analysis of a given OR-join does not consider the possibility of a token arriving from a path which has an OR-join, this could result in premature enabling and multiple execution of OR-join tasks when they are nested.

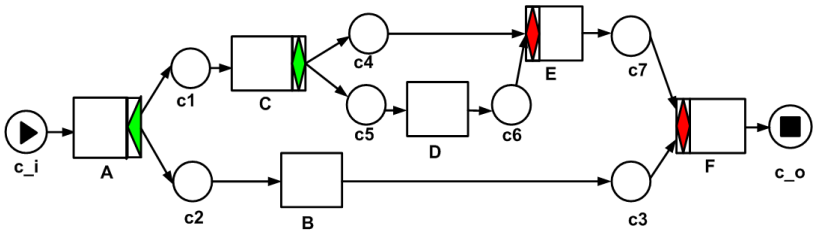

Fig. 5. A YAWL net with two OR-join tasks E and F

The other problem is that the OR-join semantics in [3] does not treat composite tasks as "black boxes", i.e., the semantics is based on the "unfolding" of the YAWL model. This semantics implies that a YAWL net at a lower level cannot be considered as a black box, thus impacting the OR-join analysis at a higher level net. Consider a specification where task B in Figure 5 is a composite task with an OR-join. When evaluating whether an OR-join should be enabled at a given marking, the analysis will be performed at lower level nets that make up a YAWL specification. This also applies for composite tasks which can deadlock. Consider marking $c2 + c7$. If task B contains a subprocess that will deadlock, then F is enabled. If B has proper completion, then F is not enabled. This also demonstrates that in the current semantics, composite tasks cannot be treated as black boxes.

2.3 Optimistic and pessimistic approaches

Instead of ignoring other OR-join tasks altogether during the analysis, we propose two alternative treatments for those OR-joins: treat them either as XOR-joins (*optimistic*) or as AND-joins (*pessimistic*). Both optimistic and pessimistic approaches achieve the

desired behaviour for an OR-join analysis by delaying enablement when there is a possibility of more tokens arriving to unmarked input conditions of the OR-join. We believe that these two alternatives result in an analysis which is more closely related to the informal semantics of OR-joins and still allow for sound semantics (i.e., avoid the fixpoint problems discussed in [2]).

The treatment of an OR-join on the path to another OR-join as an XOR-join is an *optimistic* approach. Consider a marking $M = c1 + c3$ in Figure 5 where an OR-join analysis for task F would be performed. Instead of ignoring the other OR-join task E during the analysis, task E will be treated as an XOR-join task. This will mean that the occurrence sequence $c1 + c3 \rightarrow^C c3 + c4 \rightarrow^E c3 + c7$ would be considered. As a result, task F is not enabled at M. This interpretation of OR-join task E as an XOR-join, prevents F from being enabled prematurely and it matches more closely with the informal semantics of OR-joins.

The treatment of an OR-join on the path to another OR-join as an AND-join is a *pessimistic* approach, as this now requires tokens in all input conditions of the AND-join before enabling. Consider again $M = c1 + c3$ in Figure 5 where an OR-join analysis for task F would be performed. This time, instead of ignoring task E, it will be treated as an AND-join task. Due to the OR-split behaviour of task C, tokens can be present in $c4$ or $c5$ or both after firing C. This occurrence sequence $c1 + c3 \rightarrow^C c3+c4+c5 \rightarrow^D c3+c4+c6 \rightarrow^E c3+c7$ is possible. As a token can be put in $c7$ while $c3$ remains marked, F is not enabled at M. This preserves the same informal semantics as an optimistic approach, and both approaches result in delaying the enablement of the OR-join task F.

In some cases, we observe that treating other OR-joins on the path as XOR-joins using an optimistic approach is more appropriate for the analysis. Consider a scenario where task C in Figure 5 is an XOR-split task rather than the OR-split task. Let us consider a marking $c1 + c3$ and that we treat task E as an AND-join task. As it is not possible for task E to fire due to the XOR-split and AND-join combination, the OR-join analysis will conclude that F should be enabled. As a result, task F could be executed more than once and the YAWL net does not have proper completion. The analysis will reach the same conclusion as the current semantics in YAWL where the semantics ignores the OR-join dependencies.

We have also found that when OR-joins are in conflict, there might not be a satisfactory treatment for OR-joins. Let N be a YAWL net and o_1, o_2 be two OR-join tasks. We define o_1 and o_2 to be *in conflict* iff o_1 is on a directed path to o_2 and o_2 is on a directed path to o_1. We have in Figure 6 an unusual situation described as a vicious circle in [15] where the OR-joins are in conflict and it is unclear what the exact informal semantics of the model should be. In Figure 6, there are two OR-join tasks B and C which are in conflict with each other. Condition $c3$ is an output condition of C and an input condition of B and $c4$ is an output condition of B and an input condition of C. Figure 6 is inspired by [15]. Consider a marking $c1 + c2$ where an OR-join analysis is carried out for task B and C. Using the *optimistic* approach, we treat task C as an XOR-join task during the analysis for B. As a result, we can find a reachable marking $c1 + c3 + c6$, which marks both input conditions of B. Therefore, B should not be enabled at $c1 + c2$. Similarly, we will treat B as an XOR-join task for the analysis of task C and there is a reachable

marking $c2 + c4 + c5$. Therefore, task C should not be enabled at $c1 + c2$. As a result of this *optimistic* approach, the YAWL net will *deadlock* because of the OR-join semantics using the *optimistic* approach. Using the *pessimistic* approach, we treat task C as an AND-join task during the analysis for B. At the marking $c1 + c2$, it is not possible to enable C due to the AND-join semantics, and therefore, task B will be enabled and can be fired. This will enable task C and after firing C, tokens will be placed in $c3$ and $c6$. Therefore, tasks B and C could potentially keep firing alternatingly thus resulting in a potentially infinite number of firings of task D. The same is true for the analysis of task C. We can see that the *pessimistic* approach would also result in improper completion. The original semantics that ignores other OR-joins would also result in a similar behaviour to the *pessimistic* approach. In this case, all three approaches deviate from the informal semantics of the OR-join and it is not possible to define the formal semantics accurately.

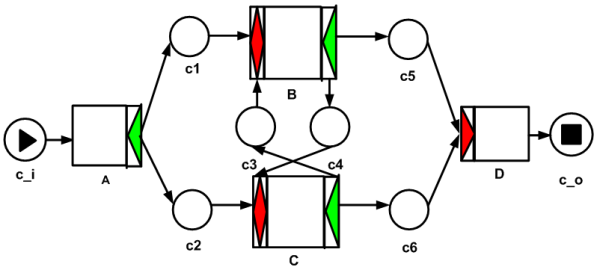

Fig. 6. OR-join tasks B and C in conflict

From the above discussions, it can be seen that there is no ideal treatment for non-local OR-join semantics in YAWL. Any formal semantics will impose some restrictions or deviate from the informal semantics to some extent. In our opinion, the XOR-join treatment of other OR-joins matches more closely the informal semantics of the OR-join. Consider the YAWL example in Figure 5 with a marking $M = c3 + c4$. If we treat E as an XOR-join during the analysis for task F, the outcome would be that F is not enabled at M because it is possible to reach a marking $M' = c3 + c7$ by executing E first. On the other hand, AND-join treatment of E will result in F being enabled at M and could result in F being executed twice. Hence, we chose to use the optimistic approach (XOR-join treatment) for our formal semantics.

3 Establishing a formal foundation

The formal semantics of YAWL is expressed in terms of a transition system [3] and while inspired by Petri nets, YAWL should not be seen as an extension of these. New concepts were introduced in YAWL to suitably deal with the workflow patterns [4]. YAWL constructs such as OR-join, cancellation and multiple instances are not directly supported by Petri nets. To perform an OR-join analysis, a multiple instances task does not effect the analysis but cancellation plays an important role (as shown in Figure 4). This cancellation feature of YAWL is theoretically closely related to Reset nets, which

are Petri nets with reset arcs. For an OR-join analysis, we propose to map a YAWL model represented as an EWF-net (Extended Workflow Net) to a Reset net. In this section, we first present the definitions of EWF-nets and then discuss the proposed abstractions to the EWF-nets. We then present the definition and firing rules for Reset nets.

3.1 EWF-nets

A YAWL model is formally defined as a nested collection of EWF-nets [3]. As we will show later, it suffices to consider only one EWF-net in isolation when evaluating an OR-join.

Definition 1 (EWF-net [3]). *An extended workflow net (EWF-net) N is a tuple* $(C, \mathbf{i}, \mathbf{o}, T, F, split, join, rem, nofi)$ *such that*[2]

- C *is a set of conditions and* T *is a set of tasks,*
- $\mathbf{i} \in C$ *is the unique input condition and* $\mathbf{o} \in C$ *is the unique output condition,*
- $F \subseteq (C \setminus \{\mathbf{o}\} \times T) \cup (T \times C \setminus \{\mathbf{i}\}) \cup (T \times T)$ *is the flow relation,*
- *every node in the graph* $(C \cup T, F)$ *is on a directed path from i to o,*
- *split:* $T \rightarrow \{AND, XOR, OR\}$ *specifies the split behaviour of each task and join:* $T \rightarrow \{AND, XOR, OR\}$ *specifies the join behaviour of each task,*
- *rem:* $T \nrightarrow \mathbb{P}(T \cup C \setminus \{\boldsymbol{i}, \boldsymbol{o}\})$ *specifies the additional tokens to be removed by emptying a part of the workflow;*
- *nofi:* $T \nrightarrow \mathbb{N} \times \mathbb{N}^{inf} \times \mathbb{N}^{inf} \times \{$*dynamic, static*$\}$ *specifies the multiplicity of each task (minimum, maximum, threshold for continuation, and dynamic/static creation of instances).*

In an EWF-net, it is possible for two tasks to have a direct connection. We will add an implicit condition $c_{(t_1,t_2)}$ between two tasks t_1, t_2 if there is a direct connection from t_1 to t_2. We denote as C^{ext} the set of conditions extended to include implicit conditions, and denote the extended flow relation as F^{ext}. We now define an explicit extended workflow net (E2WF-net) using C^{ext} and F^{ext} as follows:

Definition 2 (E2WF-net). *Let* $N = (C, \mathbf{i}, \mathbf{o}, T, F, split, join, rem, nofi)$ *be an EWF-net, the corresponding explicit EWF-net (E2WF-net) is defined as* $(C^{ext}, \mathbf{i}, \mathbf{o}, T, F^{ext}, split, join, rem, nofi)$ *where*

$$
\begin{aligned}
C^{ext} &= C \cup \{c_{(t_1,t_2)} \mid (t_1, t_2) \in F \cap (T \times T)\} \text{ and} \\
F^{ext} &= (F \setminus (T \times T)) \\
&\quad \cup \{(t_1, c_{(t_1,t_2)}) \mid (t_1, t_2) \in F \cap (T \times T)\} \\
&\quad \cup \{(c_{(t_1,t_2)}, t_2) \mid (t_1, t_2) \in F \cap (T \times T)\}.
\end{aligned}
$$

Let N be an E2WF-net and $x \in C^{ext} \cup T$, we use $\bullet x$ and $x\bullet$ to denote the set of inputs and outputs of a node i.e. $\bullet x = \{y | (y, x) \in F^{ext}\}$ and $x\bullet = \{y | (x, y) \in F^{ext}\}$. A marking is denoted by M and, just as with ordinary Petri nets, it can be interpreted as a vector, function, and multiset. M is an m-vector, where m is the total number

[2] Note that we are using basic mathematical notations such as $\nrightarrow$ for a partial function, $\mathbb{P}$ for powerset, $\mathbb{N}$ for natural numbers, and $\mathbb{N}^{inf}$ for $\mathbb{N} \cup \{inf\}$.

of conditions. This vector can also be seen as a function $M : C^{ext} \rightarrow \mathbb{N}$, where $M(c)$ returns the number of tokens in a condition c of a marking M. Functions mapping some domain (in this case C) onto $\mathbb{N}$ can also be seen as multisets, i.e., M is a multiset over C. Since a marking is a multiset, we can use notations such as $M \leq M'$, $M + M'$, and $M - M'$. $M \leq M'$ iff $\forall_{c \in dom(M)} M(c) \leq M'(c)$. $M + M'$ and $M - M'$ are a multisets such that for any $c \in dom(M)$: $(M + M')(c) = M(c) + M'(c)$ and $(M - M')(c) = M(c) - M'(c)$.

Tasks are the active components of an E2WF-net and when a task t fires at a marking M, it changes the state and reaches a new marking M', denoted as $M \rightarrow^t M'$. A YAWL specification supports hierarchy and a composite task is mapped onto an EWF-net. As we will abstract from composition, we refer the reader to [3] for a formal definition of a YAWL specification.

3.2 Abstractions

We propose to abstract the constructs in YAWL that do not affect an OR-join analysis. They include multiple instances, composite tasks and internal conditions of a task. We can assume that if a multiple instances task is enabled and executed, it will complete and put tokens into the appropriate output conditions of the task. Similarly, with the state transitions and internal conditions within a task, we can abstract from these transitions and only consider the input and output conditions of a task. In the mappings to Reset nets, we will introduce one place for each task which indicates whether a task is currently executing and as a result, abstract from the internal conditions of a task. We also propose to treat EWF-nets as *flat* nets, and ignore the hierarchical structure for the purpose of an OR-join analysis. In other words, when deciding whether an OR-join should be enabled at a given marking, we will not be considering the effect of deadlock within a composite task. We assume that a YAWL subnet which is used as a composite task at a given level is sound. Therefore, if a composite task can be enabled and executed, it will terminate at some time, and tokens will be placed in the appropriate output condition(s) of the composite task. As a result, even if there is an OR-join task in the composite task, it will not influence the decision to enable an OR-join task at a higher level. We recognise that due to the semantics of only considering tasks at the same level, the OR-join task could wait and result in a deadlock if a composite task is not sound and could deadlock. Because of these proposed abstractions from an EWF-net, we are now able to map to a Petri net like formalism. During an OR-join analysis, we are only required to consider the split and join behaviours of tasks and the cancellation set that is associated with a task. To support the cancellation feature of an EWF-net, we propose to map an EWF-net onto a Reset net.

3.3 Reset nets

A Reset net is a Petri net with special reset arcs, that can clear the tokens in selected places. Reset arcs do not change the requirements of enabling a transition but when a transition fires, they will remove tokens from the specified places. The reset arcs are used to underpin the *rem* function that models the cancellation feature of EWF-nets, cf.

Definition 1. This approach allows us to leverage existing literature and techniques in the area of Petri nets and Reset nets in particular [5–8, 10–12].

Definition 3 (Reset net). *A Petri net is a tuple (P, T, F) where P is a set of places, T is a set of transitions, $P \cap T = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$. A Reset net is a tuple (P, T, F, R) where (P, T, F) is a Petri net and $R \in T \nrightarrow \mathbb{P}(P)$ is the set of reset arcs associated with every transition $t \in T$.*

In the remainder of the paper, when we use the expression $F(x, y)$, it denotes 1 if $(x, y) \in F$ and 0 if $(x, y) \notin F$. A reachable marking M' is defined by first removing tokens needed for enabling t from its input places ($\bullet t$), then removing all tokens from reset places and then finally adding tokens to the output places of t ($t\bullet$). The notation $M[P]$ denotes function restriction and restricts M to a set of places P, i.e., a projection.

Definition 4 (Enabling and firing Reset nets). *Let (P, T, F, R, M) be a marked Reset net. A transition $t \in T$ is enabled iff $\bullet t \leq M$. Firing t at marking M reaches marking M', denoted by $M \rightarrow^t M'$, iff $\bullet t \leq M$ and $M' = (M - \bullet t)[P \setminus R(t)] + t\bullet$.*

Definition 5 (Occurrence sequence). *Let $((P, T, F, R), M_0)$ be a marked Reset net. Let $M_1, ..., M_n$ be markings of the reset net and let $t_0, t_1, ..., t_{n-1}$ be transitions in T. Sequence $s = M_0 t_0 M_1 ... t_{n-1} M_n$ is an occurrence sequence iff $M_i \rightarrow^{t_i} M_{i+1}$ for all $i, 0 \leq i \leq n-1$. A marking M'' is reachable from a marking M, written $M \rightarrow^* M''$, iff there is an occurrence sequence with initial marking M and final/last marking M''.*

To conclude this section, we define the notion of backward firing. This notion will be used to analyze coverability and is required for the OR-join analysis as is described in the remainder of this paper.

Definition 6 (Backward firing). *Let (P, T, F, R) be a Reset net and let M and M' be a markings of this net. $M' \dashrightarrow^t M$ if and only if it possible to fire a transition t backwards starting from M and resulting in M'.* [3]

$$M' \dashrightarrow^t M \Leftrightarrow \forall p \in R(t) : M(p) \leq F(t,p) \wedge$$
$$M'(p) = \begin{cases} (M(p) \dot{-} F(t,p)) + F(p,t) & \text{if} \quad p \in P \setminus R(t) \\ F(p,t) & \text{if} \quad p \in R(t). \end{cases}$$

For any reset place p, $M(p) \leq F(t, p)$ because it is emptied when firing and then $F(t, p)$ tokens are added. We do not require $M(p) = F(t, p)$ because the aim is coverability and not reachability. M', i.e., the marking before (forward) firing t, should at least contain the *minimal* number of tokens required for enabling and resulting in a marking of at least M. Therefore, only $F(p, t)$ tokens are assumed to be present in a reset place p.

4 Linking YAWL to Reset nets

In this section, we describe how an EWF-net could be transformed into a Reset net. After the abstractions from multiple instances, composite tasks and internal places in

[3] For any natural numbers a, b: $a \dot{-} b$ is defined as $\max(a - b, 0)$.

a YAWL net, we can consider a YAWL net as having tasks with various split and join behaviours and possible cancellation sets and explicit and implicit conditions. For an EWF-net without OR-join tasks, there is then a straight-forward mapping into a Reset net. For an EWF-net with OR-join tasks, we propose to use the *optimistic* treatment whereby other OR-joins on the path are replaced with XOR-joins, and perform the necessary transformations.

4.1 Semantics of an EWF-net without OR-joins

For every task t in an E2WF-net, we split t into t_{start} and t_{end} to support the various split and join constructs in YAWL. The number of t_{start} transitions depends on the join behaviour of a task and the number of t_{end} transitions depends on the split behaviour. Figure 7 illustrates the approach taken in the transformation.

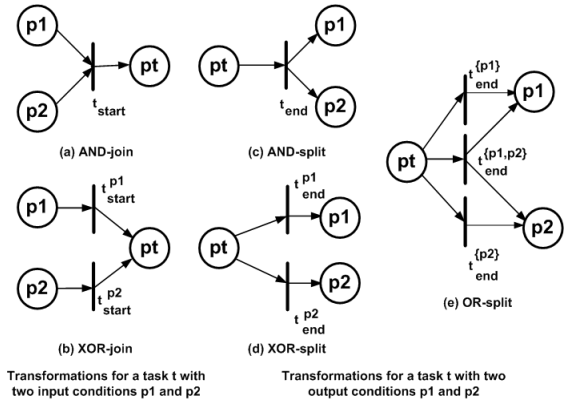

Fig. 7. Reset net transformations for YAWL split and join behaviours

Definition 7 (E2WF-Reset net). *Let* $N = (C^{ext}, \mathbf{i}, \mathbf{o}, T, F^{ext}, split, join, rem, nofi)$ *be an E2WF-net without OR-joins. A corresponding E2WF-Reset net is a tuple* (P, T', F', R) *such that*

$P = C^{ext} \cup \{p_t | t \in T\}$ *is a set of places,*
$T' = T_{start} \cup T_{end}$ *such that*
$T_{start} = \{t_{start} | t \in T \ \wedge \ join(t) = AND\}$
$\quad \cup \{t^p_{start} | t \in T \ \wedge \ join(t) = XOR \ \wedge \ p \in \bullet t\}$,
$T_{end} = \{t_{end} | t \in T \ \wedge \ split(t) = AND\}$
$\quad \cup \{t^p_{end} | t \in T \ \wedge \ split(t) = XOR \ \wedge \ p \in t\bullet\}$
$\quad \cup \{t^x_{end} | t \in T \ \wedge \ split(t) = OR \ \wedge \ x \subseteq t\bullet \wedge x \neq \emptyset\}$,
$F' = \{(p, t_{start}) | t \in T \ \wedge \ join(t) = AND \ \wedge \ p \in \bullet t\}$
$\quad \cup \{(t_{start}, p_t) | t \in T \ \wedge \ join(t) = AND\}$
$\quad \cup \{(p_t, t_{end}) | t \in T \ \wedge \ split(t) = AND\}$
$\quad \cup \{(t_{end}, p) | t \in T \ \wedge \ split(t) = AND \ \wedge \ p \in t\bullet\}$
$\quad \cup \{(p, t^p_{start}) | t \in T \ \wedge \ join(t) = XOR \ \wedge \ p \in \bullet t\}$

$\cup\{(t^p_{start}, p_t) | t \in T \ \wedge \ join(t) = XOR \ \wedge \ p \in \bullet t\}$
$\cup\{(p_t, t^p_{end}) | t \in T \ \wedge \ split(t) = XOR \ \wedge \ p \in t\bullet\}$
$\cup\{(t^p_{end}, p) | t \in T \ \wedge \ split(t) = XOR \ \wedge \ p \in t\bullet\}$
$\cup\{(p_t, t^x_{end}) | t \in T \ \wedge \ split(t) = OR \ \wedge \ x \subseteq t\bullet \ \wedge x \neq \emptyset\}$
$\cup\{(t^x_{end}, p) | t \in T \ \wedge \ split(t) = OR \ \wedge \ p \in x \ \wedge \ x \subseteq t\bullet \ \wedge x \neq \emptyset\}$,
$R \in T' \nrightarrow \mathbb{P}(P)$ *and* $dom(R) \subseteq T_{end}$ *such that*
$t \in T \wedge split(t) = AND$
$\Rightarrow R(t_{end}) = \{p_{t'} | t' \in rem(t) \cap T\} \cup (rem(t) \cap C^{ext})$,
$t \in T \wedge split(t) = XOR \ \wedge \ p \in t\bullet$
$\Rightarrow R(t^p_{end}) = \{p_{t'} | t' \in rem(t) \cap T\} \cup (rem(t) \cap C^{ext})$,
$t \in T \wedge x \subseteq t\bullet \ \wedge x \neq \emptyset \ \wedge \ split(t) = OR$
$\Rightarrow R(t^x_{end}) = \{p_{t'} | t' \in rem(t) \cap T\} \cup (rem(t) \cap C^{ext})$.

The set of reset places for a given transition t_{end} has been defined in R to support the cancellation feature in YAWL. A place p_t is also introduced to represent an internal place between t_{start} and t_{end}. The flow relation F' is also modified so that the newly introduced places in P and transitions T' are properly connected.

The function *marked* returns the set of marked conditions in an EWF-net for a given marking M.

Definition 8 (Marked). *For a marking M of an E2WF-Reset net:*
marked(M)=$\{c \in dom(M) \mid M(c) > 0\}$.

The $\sqsubseteq$ relation indicates that M marks fewer or the same places as M'. This is a looser notion of smaller markings than $\leq$, because only the marking of places is considered and the number of tokens in a place is ignored. The notation $\sqsubset$ is used to indicate that M marks strictly less places than M'. The notation $M[C]$ restricts M to a set of conditions C, i.e., a projection. For instance, $M[t\bullet] \sqsubset M'[t\bullet]$, represents a comparison between M and M' that is restricted to the output places of t.

Definition 9 ($\sqsubseteq$). *Let M, M' be two markings of an E2WF-Reset net and C a set of conditions: $M \sqsubseteq M'$ iff marked(M) $\subseteq$ marked(M'), $M \sqsubset M'$ iff $M \sqsubseteq M'$ and $\neg(M' \sqsubseteq M)$.*

We now define how a given marking M in an E2WF-net can be linked to a marking M^* in the corresponding E2WF-Reset net. For all the conditions that exist in an E2WF-net, they will be marked exactly the same in M^* and zero tokens for the newly introduced places in the E2WF-Reset net i.e. $M = M^*$.

Definition 10 (M^*). *Let (N, M) be a marked E2WF-net and N^* be the corresponding marked E2WF-Reset net of N, then M corresponds in a natural way to a marking of N^*. This marking marks all the places in N^* which correspond to conditions in N with the same number of tokens. We will refer to this as the corresponding marking and denote it as M^*.*

We define the enabling and firing rules for tasks in an E2WF-net using the transition firing rules as defined for Reset nets. Executing a task of an E2WF-net corresponds to executing the corresponding start and end transitions t_{start} and t_{end} of the E2WF-Reset net.

Definition 11 (Enabling and firing E2WF-net). *Let (N, M) be a marked E2WF-net and (N^*, M^*) be the corresponding marked E2WF-Reset net. A task t is enabled at (N, M) iff $\bullet t \leq M^*$. Firing t at M reaches M', denoted by $M \rightarrow^t M'$ iff for the corresponding start transition t_{start} and end transition t_{end}, we have $M^* \rightarrow^{t_{start}} M'' \rightarrow^{t_{end}} M^{*'}$.*

Note that this definition allows us to transfer typical Petri-net concepts such as reachability to E2WF-nets.

We are seeking a predicate *superM* to determine whether we can reach a marking that marks more places than M for a certain set of places. From a given marking M and a given set of places P', we can determine whether it is possible to reach a marking from M which marks more places in P'. If we define $P' = \bullet$*o-j*, a set of input conditions of an OR-join, then we can determine whether a bigger marking (restricted to places in P') exists for a given marking M (in which case the OR-join is not enabled).

Definition 12 (superM). *Let $N = (P, T, F, R, M)$ be a marked E2WF-net and $P' \subseteq P$ be a set of places for consideration, superM(N, M, P') holds iff there is a marking M' such that $M \rightarrow^* M'$ and $M[P'] \sqsubset M'[P']$.*

4.2 Semantics of an EWF-net with OR-joins

The transformation from an EWF-net with OR-join tasks into an E2WF-OJ is identical to E2WF-Reset net transformation for all tasks that are not OR-join tasks. The additional steps to incorporate OR-join tasks are include creating a set OJ for the t_{start} transition of each OR-join task in the E2WF-net and adding t_{start} transitions in OJ into T_{start}.

Definition 13 (E2WF-OJ). *Let N be an EWF-net with OR-joins and N^{ext} be the E2WF-net of N, the corresponding E2WF-OJ is a tuple (P, T'', F'', R, OJ) such that P, T', T_{start}, T_{end}, F', and R are as defined in Definition 7 and T'', F'', OJ are defined as follows:*

$T'' = T'_{start} \cup T_{end}$,
$T'_{start} = T_{start} \cup \{t_{start} | t \in T \wedge join(t) = OR)\}$,
$F'' = F' \cup \{(p, t_{start}) | t \in T \wedge join(t) = OR \wedge p \in \bullet t\} \cup$
$\{(t_{start}, p_t) | t \in T \wedge join(t) = OR\}$, *and*
$OJ = \{t_{start} | t \in T \wedge join(t) = OR\}$.

The function OJ-Remove is used to transform E2WF-OJ by replacing the join behaviour of all the OR-join tasks in an E2WF-net to XOR-join and removing the OR-join task in question. This effectively converts an E2WF-OJ into an E2WF-Reset net so that we can use the transition firing rules and superM predicate defined for Reset nets.

Definition 14 (OJ-Remove function). *Let $N' = (P, T, F, R, OJ)$ be an E2WF-OJ for an EWF-net N and $j \in OJ$ be an OR-join task under consideration. The function OJ-Remove(N', j) returns (P', T', F', R') such that*

$P' = P$,
$T' = (T \setminus OJ) \cup \{t^p_{start} | t \in OJ \setminus \{j\} \wedge p \in \bullet^N t\}$,

$$
\begin{aligned}
F' =& F \cap ((P' \times T') \cup (T' \times P')) \\
& \cup \{(p, t^p_{start}) | p \in {\bullet^N} t \ \wedge \ t \in OJ \setminus \{j\}\} \\
& \cup \{(t^p_{start}, p_t) | p \in {\bullet^N} t \ \wedge \ t \in OJ \setminus \{j\}\} \\
R' =& R.
\end{aligned}
$$

The firing rules for an E2WF-OJ are defined as follows. The firing rule for a transition t which is not an OR-join is the same as for Reset nets. For transitions *o-j* that are OR-joins in E2WF-net, (i.e. *o-j* $\in OJ$), the firing rule is defined in two steps. We first use the OJ-Remove function to transform other OR-joins (except *o-j*) into XOR-joins and produce an equivalent E2WF-Reset net. We then check whether *superM* holds. If *superM* holds then the OR-join, *o-j*, should not be enabled at M. Otherwise, *o-j* is enabled at M.

Definition 15 (Enabling rule). *Let (P, T, F, R, OJ, M) be a marked E2WF-OJ. A transition $t \in T \setminus OJ$ is enabled at M iff ${\bullet}t \leq M$. A transition o-j $\in OJ$ is enabled at marking M iff at least one of its input places is marked and superM(OJ-Remove$(P, T, F, R, OJ,$ o-j$), M, {\bullet}$o-j$)$ does not hold.*

Definition 16 (Forward firing). *When a transition t of an E2WF-OJ is enabled at a marking M', it can fire and a new marking M is reached.*

$$
M' \rightarrow^t M \Leftrightarrow \forall p \in P : M'(p) \geq F(p,t) \wedge
$$
$$
M(p) = \begin{cases} M'(p) - F(p,t) + F(t,p) & \text{if} \quad p \in P \setminus R(t) \\ F(t,p) & \text{if} \quad p \in R(t). \end{cases}
$$

Definition 17 (Reachable markings). *We denote $M \rightarrow M'$ iff there is a $t \in T$ such that $M \rightarrow^t M'$. We denote $M \rightarrow^* M''$ if there is an occurrence sequence from M to M''.*

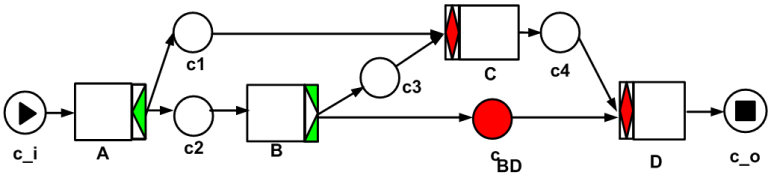

Fig. 8. An E2WF-net N with OR-join tasks C and D

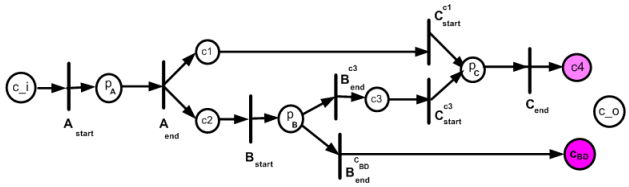

Fig. 9. An E2WF-Reset net for OR-join analysis of task D in Figure 8

We will now describe how the transformations will be performed for an EWF-net with two OR-join tasks C and D as shown in Figure 8. The shaded place indicates the

explicit condition c_{BD} which has been added for the implicit condition between tasks B and D. Figure 9 shows an equivalent Reset-net for the E2WF-net in Figure 8 for OR-join analysis of task D. The OR-join task C is on the path to task D and the OJ-Remove function is applied to treat task C as an XOR-join task. Also note that OR-join task D has been removed from the net by the OJ-Remove function.

Consider a marking $M = c1+c_{BD}$ of N where OR-join analysis for task D would be performed. The input places of task D are $c4$ and c_{BD}. We need to investigate whether it is possible to reach a marking that marks both $c4$ and c_{BD}. We can observe the sequence $c1 + c_{BD} \rightarrow^{C^{c1}_{start}} p_C + c_{BD} \rightarrow^{C_{end}} c4 + c_{BD}$ exists and that we can reach $M' = c4 + c_{BD}$ from M. Therefore, ***superM*** predicate holds as $M \rightarrow^* M'$ and $M[\{c4, c_{BD}\}] \sqsubset M'[\{c4, c_{BD}\}]$. The OR-join analysis for task D will conclude that D should not be enabled at marking M as it is possible to reach a marking from M that marks more input places of the OR-join than M does.

5 OR-join algorithm proposal

The main objective of the OR-join algorithm is to determine, for a given OR-join, whether or not a marking M' is reachable from a given marking M that marks more input places of that OR-join exists. We perform this analysis by first transforming an EWF-net (with OR-joins) into an E2WF-Reset net for a given OR-join task and then by calling the OR-join algorithm. Our algorithm is based on backward search techniques for Well-Structured Transition Systems (WSTSs) [5, 7, 10–12]. The algorithm works backwards by computing the predecessor markings for a given marking, as opposed to the forward approach used in coverability tree algorithms. A Reset net can be represented as a WSTS and the backwards algorithm has been successfully applied to solve the coverability problems for Reset nets [7, 16].

5.1 Backward algorithm for OR-join analysis

WSTSs are "a general class of infinite state systems for which decidability results rely on the existence of a well-quasi-ordering between states that is compatible with the transitions." [12]. The existence of a well-quasi-ordering over an infinite set of states ensures the decidability of termination and coverability properties [7, 12].

Definition 18 (Well-Structured Transition System [7]). *A well-structured transition system (WSTS) is a structure $S = \langle Q, \rightarrow, \leq \rangle$ such that $Q = \{m, ...\}$ is a set of states, $\rightarrow \subseteq Q \times Q$ is a set of transitions, $\leq \subseteq Q \times Q$ is a well-quasi-ordering (wqo) on the set of states, satisfying the simple monotonicity property, $m \rightarrow m'$ and $m_1 \geq m$ imply $m_1 \rightarrow m'_1$ for some $m'_1 \geq m'$.*

Reset nets can be seen as a WSTS $\langle Q, \rightarrow, \leq \rangle$ with Q the set of markings, $M \rightarrow M'$ if for some t, we have $M \rightarrow^t M'$ and $\leq$ the corresponding $\leq$ order on markings (which is a wqo) [16].

Definition 19 (Upward-closed set [12]). *Given a quasi-ordering $\leq$ on X, an upward-closed set is any set $I \subseteq X$ such that $y \geq x$ and $x \in I$ entail $y \in I$. To any $x \in X$*

we associate $\uparrow x =^{def} \{y | y \geq x\}$. *A basis of an upward-closed* I *is a set* I^b *such that* $I = \bigcup_{x \in I^b} \uparrow x$.

Given a WSTS $\langle Q, \rightarrow, \leq \rangle$ and a set of states $I \subseteq Q$, $Pred(I)$, $pb(I)$ and $Pred^*(I)$ can be defined [16]. The immediate predecessors of I: $Pred(I) = \{x | x \rightarrow y \wedge y \in I\}$, all predecessor states of I, $Pred^*(I) = \{x | x \rightarrow^* y \wedge y \in I\}$ and $pb(I) = \bigcup_{y \in I} pb(y)$ where $pb(y)$ yields a finite basis of $\uparrow Pred(\uparrow\{y\})$ (i.e., $pb(y)$ yields a finite set such that $\uparrow pb(y) = \uparrow Pred(\uparrow\{y\})$) [16]. The coverability problem for a Reset net is as follows: given two markings x and y can we reach $y' \geq y$ starting from x [16]. Provided that $\leq$ is decidable and $pb(y)$ exists and can be effectively computed [12], the backwards reachability analysis can be performed to decide the coverability [7, 10, 16]. $\{y\}$ is a basis of upward closed set $\uparrow\{y\}$ and we can determine that y is coverable from x if there exists a $x' \in Pred^*(\uparrow\{y\})$ such that $x' \leq x$ (because $\leq$ is a wqo). As $\uparrow\{y\}$ is upward-closed, $Pred^*(\uparrow\{y\})$ is upward-closed [12]. We can compute a finite basis of $Pred^*(\uparrow\{y\})$ as the limit of the sequence $I_0 \subseteq I_1 \subseteq ...$ where $I_0 =^{\text{def}} \{y\}$ and $I_{n+1} =^{\text{def}} I_n \cup pb(I_n)$ [16]. The sequence eventually stabilises at some I_n when $\uparrow I_{n+1} = \uparrow I_n$ and we have reached a stabilisation point that has the property $\uparrow I_n = Pred^*(\uparrow\{y\})$ [16]. The coverability question now becomes: is there an $x' \in \uparrow I_n$ such that $x' \leq x$. I_n is a finite basis for $Pred^*(\uparrow\{y\})$ and the coverability question can now be answered by testing whether there exists a $x' \in I_n$ such that $x' \leq x$.

We now present the procedures that operationalise the coverability question for Reset nets. The procedure $\mathbf{Coverable}$ returns a Boolean value to indicate whether a marking t is coverable from a marking s of a Reset net [16].

PROCEDURE $\mathbf{Coverable}$ (Marking x, y): Boolean
Marking x';
BEGIN
  for $x' \leq x$ **do**
    if $x' \in \mathbf{FiniteBasisPred}^*(\{y\})$ **then return** TRUE; **end if**;
  end for;
  return FALSE;
END

The procedure $\mathbf{FiniteBasisPred}^*$ returns a set of markings which represents a finite basis of all predecessors and is based on the method described in [16].

PROCEDURE $\mathbf{FiniteBasisPred}^*$ (SET Marking I): SET Marking
SET Marking K, K_{next};
BEGIN
  $K := I$; $K_{next} := K \cup \mathbf{pb}(K)$;
  while not $\mathbf{IsUpwardEqual}(K, K_{next})$ **do**
    $K := K_{next}$; $K_{next} := K \cup \mathbf{pb}(K)$;
  end while;
  return K;
END

The procedure call $\mathbf{IsUpwardEqual}(K, K_{next})$ is used to detect whether the stabilisation point has been reached i.e. $\uparrow K_{next} = \uparrow K$, cf. [11]. The procedure $\mathbf{pb}(I)$ returns $pb(I)$ such that $pb(I) = \bigcup_{x \in I} pb(x)$ [16].

PROCEDURE $\mathbf{pb}$ (SET Marking I): SET Marking
Set Marking $Z = \emptyset$; Marking M;
BEGIN
  for $M \in I$ **do** $Z := Z \cup \mathbf{pb}(M)$; **end for**;
  return Z;
END

$\mathbf{pb}(M)$ is effectively computed for Reset nets by "executing the transitions backwards and setting a place to the minimum number of tokens required to fire the transition if it caused a reset on this place" [16].[4] Note that, in our case, this minimum is one as we do not have weighted arcs. We will make use of backward firing rule as defined in Definition 6. For each transition $t \in T$, we determine whether an M' exists such that $M' \dashrightarrow^t M$. Hence, $pb(M) = \{M' | \exists_{t \in T}\ M' \dashrightarrow^t M\}$.

PROCEDURE $\mathbf{pb}$ (Marking M): SET Marking
SET Marking $Z = \emptyset$;
BEGIN
  for $t \in T$ **do**
    if $M[R(t)] \leq t \bullet [R(t)]$ **then**
      $Z := Z \cup \{(M \dot{-} t \bullet + \bullet t)[P \setminus R(t)] + (M + \bullet t)[R(t)]\}$;
    end if;
  end for;
  return Z;
END

We can then apply the coverability findings of a Reset net to the OR-join analysis. Let (N, M) be a marked E2WF-net, *o-j* be the OR-join task under consideration, X be $\bullet$ *o-j*, N' be the corresponding E2WF-Reset net and Y be a set of markings such that each marking in Y has only one token in each of the marked input places of *o-j* in M and one token in exactly one of the unmarked input places of the *o-j* in M. To determine whether *o-j* should be enabled at M, we need to determine whether there exists a $M' \in \mathit{Pred}^*(M_w)$ such that $M' \leq M$ for each of the markings $M_w \in Y$ (coverability question). Each marking M_w in Y satisfies the condition $M[X] \sqsubset M_w[X]$, i.e. M_w has tokens in more input places of the OR-join *o-j* and if M_w can be reached from M, the OR-join is not enabled. The procedure $\mathbf{OrJoinEnabled}$ is called with parameters M and X and it returns a Boolean value to indicate whether the OR-join should be enabled at M.

PROCEDURE $\mathbf{OrJoinEnabled}$ (Marking M, SET Place X): Boolean
SET Marking Y; Marking M_w;

[4] Note that the algorithm described in [16] is incorrect. On Page 105 in [16], $pb(M)$ is defined in a rather naive way. Applying $pb(M)$ to the empty marking yields a counter example, since it is not a finite basis for $\uparrow \mathit{Pred}^*(\uparrow\{M\})$.

BEGIN
$Y = \{q + \sum_{p \in X : M(p) > 0} p \mid q \in X \ \wedge \ M(q) = 0\}$;
for $M_w \in Y$ **do**
if $\mathbf{Coverable}(M, M_w)$ **then return** FALSE; **end if**;
end for;
return TRUE;
END

5.2 A worked example

Throughout the paper we have shown several examples where it is a non-trivial task to decide if an OR-join is enabled or not. Clearly, the algorithm can be applied successfully to these situations. To illustrate its inner working in some detail we use one last example.

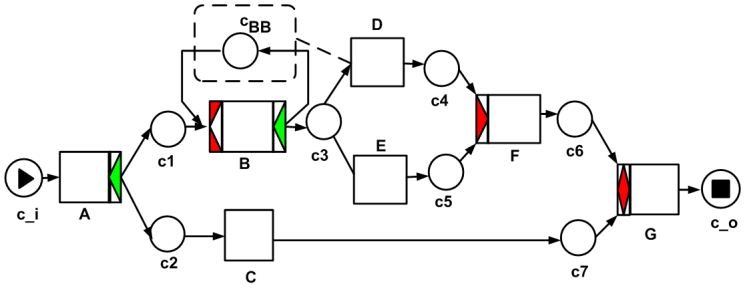

Fig. 10. A YAWL net with an OR-join task G and cancellation

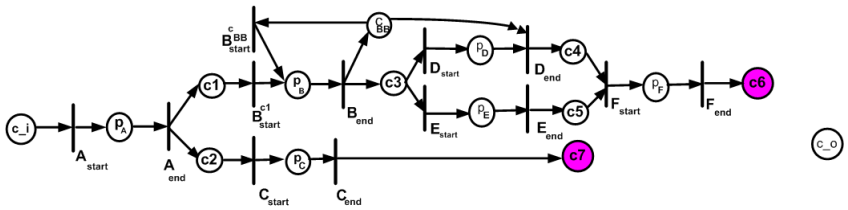

Fig. 11. A corresponding Reset net for Figure 10 (note the double-headed arrow denoting the reset arc from C_{BB} to D_{end})

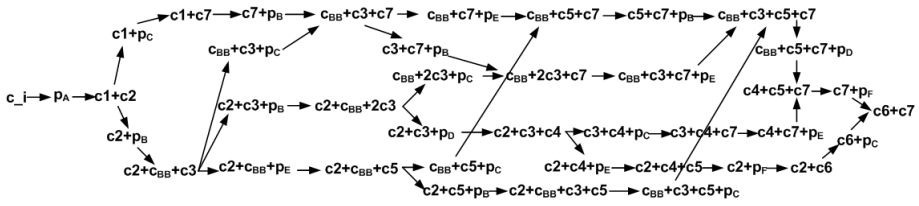

Fig. 12. Illustration of backwards reachablility analysis

Consider a marking $M = c1 + c7$ in Figure 10 where the OR-join analysis for task G is carried out. It is possible to have an occurrence sequence, $c1 + c7 \rightarrow^B c_{BB} + c3 + c7 \rightarrow^E c_{BB} + c5 + c7 \rightarrow^B c_{BB} + c3 + c5 + c7 \rightarrow^D c4 + c5 + c7 \rightarrow^F c6 + c7$. As a result, $c6 + c7$ is a reachable marking from $c1 + c7$ and the OR-join should not be enabled at marking M. The evaluation will start with a call to the procedure $\mathbf{OrJoinEnabled}(c1 + c7, \{c6, c7\})$. $Y := \{c6 + c7\}$ and for $M_w = c6 + c7$, we will obtain a finite basis of all the predecessors of $c6 + c7$. Figure 12 illustrates the backwards reachability analysis [11], with the basis of the predecessor markings for $c6 + c7$. It can be seen that $c1 + c7$ is a predecessor of $c6 + c7$. $M' \leq M$ includes the following markings $\{c1, c7, c1 + c7\}$. As $M' = c1 + c7$ is in the predecessors for $c6 + c7$, the procedure will return FALSE, concluding that the OR-join should not be enabled at M.

6 Conclusion

This paper focuses on the OR-join construct in YAWL and proposes a new semantics. The decision to enable an OR-join task cannot be made locally: an OR-join task should only be enabled when there is at least one token in one of the input conditions and there is no possibility of a token arriving at one of the yet unmarked input conditions of the OR-join. Otherwise, the OR-join task should wait for synchronisation. Instead of ignoring other OR-joins on the path, we propose two alternative approaches (optimistic or pessimistic) for OR-joins which are on the path of other OR-joins. Reset nets are used as formal basis for OR-join analysis to support cancellation features. This is made possible by the fact that we can abstract from the concepts of YAWL such as multiple instances, composite task and internal state transitions of a task. We present transformation rules from a YAWL model with OR-joins to a Reset net for a specific OR-join analysis. We then propose an OR-join evaluation algorithm which is based on the backward search techniques for Well-Structured Transition Systems. The algorithm does not yet exploit potential optimisation techniques as e.g. presented in [10].

To conclude the paper, we would like to emphasise that the results reported in this paper are not limited to YAWL. As is indicated in the introduction, many workflow management systems, but also other process-aware information systems (e.g., ERP, CRM, and PDM systems), have problems dealing with the OR-join. In fact, the problem surfaces in many other domains [19].

Acknowledgements. We would like to especially thank Philippe Schnoebelen and Jerome Leroux for their valuable input on the issue of decidability of OR-join algorithm and for many useful references provided in the area of Reset nets.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P van der Aalst, J. Desel, and E. Kindler. On the Semantics of EPCs: A Vicious Circle. In M. Rump and F.J. Nüttgens, editors, *Proceedings of the EPK 2002: Business Process*

Management using EPCs, pages 71–80, Trier, Germany, 2002. Gesellschaft für Informatik, Bonn.

3. W.M.P. van der Aalst and A.H.M ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, June 2005.
4. W.M.P van der Aalst, A.H.M ter Hofstede, B.Kiepuszewski, and A.P.Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14:5–51, 2003.
5. P.A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (27 - 30 July)*, pages 313–321, New Brunswick, NJ, July 1996. IEEE Computer Society.
6. P. Darondeau. Unbounded Petri net Synthesis. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 413–428, Eichstätt,Germany, 2003. Springer-Verlag.
7. C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset Nets Between Decidability and Undecidability. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115, Aalborg, Denmark, July 1998. Springer-Verlag.
8. C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T Nets. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Lectures on Concurrency and Petri Nets*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310, Prague, Czech Republic, July 1999. Springer-Verlag.
9. Eastman Software. *RouteBuilder Tool User's Guide*. Eastman Software, Inc, Billerica, MA, USA, 1998.
10. A. Finkel, J.-F. Raskin, M. Samuelides, and L. van Begin. Monotonic Extensions of Petri Nets: Forward and Backward Search Revisited. *Electronic Notes in Theoretical Computer Science*, 68(6):1–22, 2002.
11. A. Finkel and Ph. Schnoebelen. Fundamental Structures in Well-Structured Infinite Transition Systems. In C.L. Lucchesi and A.V. Moura, editors, *Theoretical Informatics: Third Latin American Symposium, Campinas, LATIN'98 (20 - 24 April)*, volume 1380 of *Lecture Notes in Computer Science*, pages 102–118, Campinas, Brazil, 1998. Springer-Verlag.
12. A. Finkel and Ph. Schnoebelen. Well-structured Transition Systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, April 2001.
13. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
14. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. Phd thesis, Queensland University of Technology, Brisbane, Australia, 2003.
15. E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In J. Desel, B. Pernici, and M. Weske, editors, *Proceedings of 2nd International Conference on Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97, Potsdam, Germany, 2004. Springer-Verlag.
16. M. Leuschel and H. Lehmann. Coverability of Reset Petri Nets and other Well-Structured Transition Systems by Partial Deduction. In J. Lloyd et al., editors, *Proceedings of Computational Logic 2000*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 101–115, London, UK, 2000. Springer-Verlag.
17. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
18. P. Rittgen. Modified EPCs and their Formal Semantics. Technical Report 99/19, Institute of Information Systems, University Koblenz-Landau, Koblenz, Germany, 1999.
19. A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the Models for Asynchronous Circuit Behaviour with OR Causality. *Formal Methods in System Design*, 9(3):189–233, 1996.